(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 May 2001 (03.05.2001)

PCT

(10) International Publication Number
WO 01/31456 A1

(51) International Patent Classification[7]: G06F 12/08

(21) International Application Number: PCT/US00/29881

(22) International Filing Date: 27 October 2000 (27.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/429,142 28 October 1999 (28.10.1999) US

(71) Applicant: CONNECTCOM SOLUTIONS, INC. [US/US]; 1150 Ringwood Court, San Jose, CA 95131 (US).

(72) Inventor: LAM, William; 757 Anacapa Court, Milpitas, CA 95953 (US).

(74) Agents: BALDWIN, Stephen, E. et al.; Trial & Technology Law Group, Suite 220, 545 Middlefield Road, Menlo Park, CA 94025 (US).

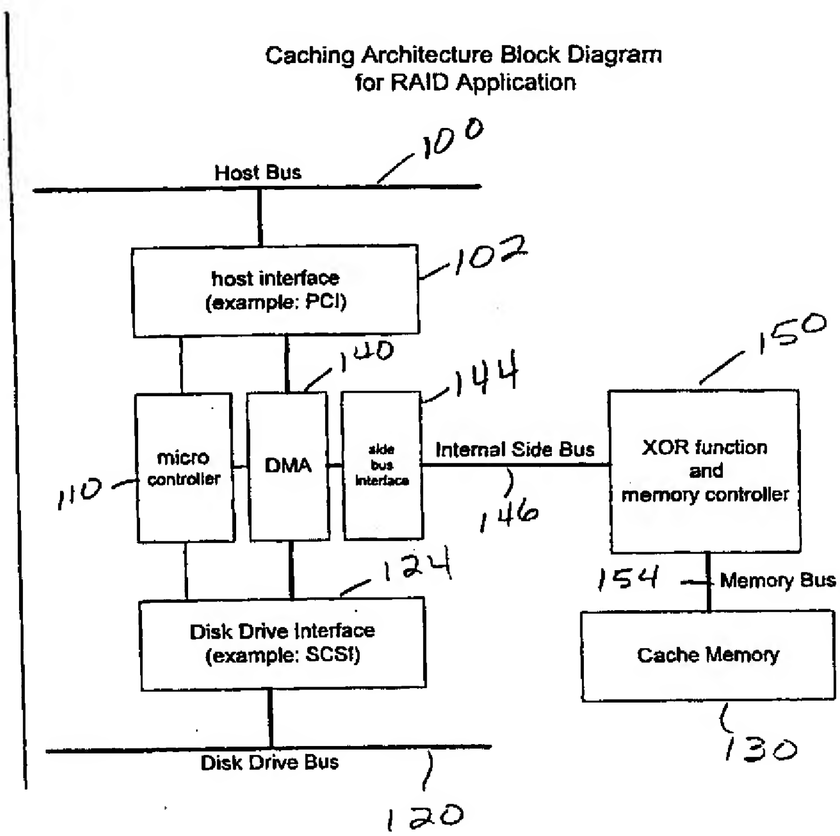
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

[Continued on next page]

(54) Title: CACHING TECHNIQUES FOR IMPROVING SYSTEM PERFORMANCE IN RAID APPLICATIONS



(57) Abstract: The present invention provides a caching technique for improving system performance in RAID (Redundant Arrays of Inexpensive Disks) applications. The memory caching architecture technique according to the present invention creates a substantial improvement on the system host bus (100), especially for a RAID application system, implementing an Exclusive-OR (XOR) function in hardware. By pulling the XOR operation from software and by implementing the XOR function in hardware (150) and by removing the unnecessary data transfer from the XOR operation, a significant improvement on the host bus bandwith can be achieved.

WO 01/31456 A1

— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Caching Techniques for Improving System Performance in RAID Applications

By:
William Lam

Background of the Invention

The present invention relates to a caching technique for improving system performance in RAID applications. The memory caching architecture technique according to the present invention creates a substantial improvement on the system host bus, especially for a RAID application system.

The present invention is related to U. S. Patent Nos. 5,734,924 entitled System For Host Accessing Local Memory, issued March 31, 1998; 5,586,268 entitled Multiple Peripheral Adapter Device Driver Architecture, issued December 17, 1996; and 5,561,813 entitled Circuit For Resolving I/O Port Address Conflicts, issued October 1, 1996, all of which are assigned to the same assignee as the present invention, and the details of which are hereby incorporated by reference.

The traditional RAID (Redundant Arrays of Inexpensive Disks) architecture is to fetch data or to store data from/to SCSI Bus (a popular disk drive Bus) to/from PCI Bus (a popular system host Bus). In many situations, the system performance would be limited by the traffic on the Host Bus. A typical data transfer size from the Host to the disk drive is in the order of 4K byte to 64K byte. When the application software needs to perform the XOR operation (from the RAID algorithm), huge amounts of data transfer required to go on the host bus. In addition, each access from the hard disk drive is very

slow relative to silicon memory access time. This in term generates a bottleneck situation on the host bus which in-effect slows down the overall system performance.

Summary of the Invention

It is an object of the present invention to provide a caching technique for improving system performance in RAID applications.

The memory caching architecture technique according to the present invention creates a substantial improvement on the system host bus, especially for the RAID application system, implementing an Exclusive-OR (XOR) function in hardware. By pulling the XOR operation from software and by implementing the XOR function in hardware and by removing the unnecessary data transfer from the XOR operation, a significant improvement on the host bus bandwidth can be achieved.

In one embodiment, the present invention provides a caching system comprising a host bus; a disk drive bus; an internal cache memory; interfacing means for interfacing between the host bus, the disk drive bus and the cache memory. The interfacing means includes a host interface for interfacing with the host bus; a disk drive interface for interfacing with the disk drive bus; XOR function means for interfacing with the cache memory and for providing and loading one or more XOR functions into the cache memory; and a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory.

In a further embodiment, the present invention includes DMA means for buffering the data transfer between the host bus, the disk drive bus and the cache memory; an internal side bus for interfacing with the XOR function means for providing direct data transfers between the host bus and the disk drive bus; and means for providing simultaneous data transfers between the host bus and the disk drive bus and the cache memory.

Other objects, features and advantages of the present invention will become apparent from the following detailed description when taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings where like numerals indicate like components and which are incorporated in and form a part of this specification, illustrate embodiments of the invention and, together with the description, serve to explain the principles of the invention:

Figure 1 shows caching architecture block diagram for the present invention.

Figure 2 shows a caching architecture block diagram with a XOR datapath and with an embedded CPU for the present invention.

Figure 3 shows an XOR function with local memory for the present invention.

Figure 4 shows a more detailed XOR function with local memory of Figure 3.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. While the invention will be described in conjunction with the preferred embodiments, it will be understood that they are not intended to limit the invention to those embodiments. On the contrary, the invention is intended to cover alternatives, modifications and equivalents, which may be included within the spirit and scope of the invention as defined by the appended claims.

Caching Technique for RAID applications

As will be described, the memory caching architecture technique according to the present invention creates a substantial improvement on the system host bus, especially for the RAID application system, implementing an exclusive-OR (XOR) function in

hardware. By pulling the XOR operation from software and by implementing the XOR function in hardware and by removing the unnecessary data transfer from the XOR operation, a significant improvement on the host bus bandwidth can be achieved.

The present invention introduces a caching technique which resides between the disk drive bus and the host bus. By having a local memory sitting between the Host Bus and the disk drive Bus, the data access time from the host can be significantly reduced when there is a cache hit or the data is in the cache memory. For the case when data is not in the cache memory, data is pulled out from the disk to the cache memory and to the host bus. A read ahead technique can be used in this case as well. More data is read out than required, which translates into more chances for the host to have a cache hit, perhaps for the next time around.

Even when the host performs a write operation to the disk, the data transfers to the cache first and then releases the host bus for other operations. This accelerates the termination of the host bus write cycle and the write transfer from the cache to the disk drive performed by the controller locally at a later time. This would isolate the host bus from the slow disk drive. The minimum size of the cache is equal to the maximum amount of each data transfer from the host multiply by the total number of the disk drives.

Since the XOR operation is extracted out from the software application, the host is no longer necessary to access the data from the slow disk drive bus to the host bus. The XOR operation can be performed behind the scene and the RAID software just dispatches the XOR operation to the bus controller. The heavy data accesses between the disk drive and the cache memory are totally isolated from the host bus. This boosts up the overall host bus performance and reduces the software time to perform the XOR operation.

Figure 1 shows the architecture described above.

Figure 1 is one example of the RAID application architecture for an embedded system. In Figure 1, the host bus interface block 102 provides the central communication between the host bus 100 and the host adapter circuitry. In most systems today, the host bus 100 would be a PCI bus. In this application, the host bus 100 communicates with the hard disk drive bus 120 through disk drive interface 124 through this architecture. This architecture provides direct data transfer from the host bus 100 to the disk drive via disk drive bus 120 or simultaneously transfer from the disk drive bus 120 to both the host bus 100 and to the cache memory 130.

This data transfer can be achieved by using the DMA datapath 140 which resides at the center of the three buses. The DMA (Direct Memory Access) datapath 140 is a media to buffer up the incoming data and to pump the data out at the appropriate time, especially since the speed from each of the three buses may not be the same. The data flow and traffic between the buses is managed by the micro-controller 110.

With the proper programming on the micro-controller 110, an optimized data flow can be precisely controlled, the caching technique can be realized, and the XOR function can be performed. The micro-controller 110 accepts the appropriate command blocks from the RAID software and the micro-controller 110 ucode would decode and execute the commands in a way such that the controls are enabling the proper datapath for a specific task. The DMA datapath 140 would provide enough data buffering on each side of the three buses, so that data transfer can be operated in an effective manner. The proper buffering size would determine by the typical transferring size, the incoming data rate and the outgoing data rate. With this architecture, the cache data can transfer to/from the host bus and the disk drive bus concurrently. This effect can be achieved from the side bus interface 144 by providing dual datapath from side bus 146 to host bus 100 and to hard disk bus 120. By alternating the transfer to this two datapath in a reasonable transfer size(such as 512B each time), a concurrent transferring effect from the software point of view can be accomplished.

One bus that DMA 140 interfaces with is the side bus 146 which communicates with the cache controller and the XOR functional block 150. This block 150 provides the capability of doing an XOR function and interfacing with the cache memory 130 via memory bus 154. This capability enables the RAID application to substantially improve the system performance. By performing the XOR function on the side while freeing up the host bus 100 for other tasks, the host bus bandwidth is improved significantly and at the same time, the time for performing the XOR task is reduced.

The side bus 146 accepts multiple XOR blocks. In this way, the expandable cache memory and multiple XOR functions can execute simultaneously. For the high-end server class system, bigger cache size and multiple XOR functions actively performing simultaneously would be desirable.

To illustrate this architecture without a microprocessor, a 64Mbyte transfer is activated from disk drive bus 120 to host bus 100 via host interface 102. With this architecture, the data transfers from the disk drive bus 120 to the host bus 100, and at the same time, the same data transfers into the cache memory 130 as well. When the host requires fetching the same set of data, the data can be fed to the host as fast as several hundreds of nanoseconds. Since the data resides in the cache memory 150, the data can be quickly accessed without going through the hard disk which normally would take up to few milliseconds. To perform the XOR function, the RAID software loads the proper source data into the cache memory 130, and the RAID software would execute the command by properly programmed the internal required registers.

The hardware puts the XOR result into the cache memory 130, and the RAID software gets the data from the cache memory 130 and put this result into the disk drive. During the XOR operation, there is no host bus transfer required, and as a result the host bus 100 is freed up during this lengthy operation.

With a microprocessor 155 embedded into the architecture as shown in Figure 1B, the top level command receives from the host and the microprocessor runs the RAID

software to decode the command and formed a list of lower level commands which usually send over the host bus 100. Now these lower level commands are sent to the micro-controller for the next level of execution through the internal bus 146. In Figure 2, the XOR function plus datapath 157 for direct cache memory 130 access includes XOR 161 and memory controller 159. This method will even further reduce the traffic on the host interface bus 102. With this embedded microprocessor capability, a complete solution for RAID application is available on a single component and many RAID systems can now be built in a cost effective manner.

Dual Data Path Providing Concurrent Cache Data Access and XOR operation

As described above, by pulling the XOR function out of the RAID application software and applying caching technique on the host read and write operations, the overall system performance improves substantially.

While the XOR hardware is executing, the data in the cache memory is locking up exclusively for this operation. The XOR operation can be very lengthy in time depending on the number of the input sources and the size of the data. If there are 8 input sources and each source is 64Kbytes, the XOR operation would occupy the cache data for few mini-seconds. This is a very long period of time. If the host bus or the disk drive bus needs to access the cache data, there is a long waiting period before the cache memory is free for access.

In accordance with a further embodiment of the present invention, the technique to improve this situation is to implement a dual datapath function 157, as shown in Figure 2. One datapath is used for the XOR operation and the other is used for direct cache memory access. While the XOR datapath 161 is performing the operation, the host bus 100 or the disk drive bus 120 can access the cache data through the second datapath. This technique would delay the XOR operation from finishing but the overall performance would improve. Since neither the host bus 100 nor the disk drive bus 120 need not have

to wait for the XOR operation 161 to complete, the RAID application software can continue to process while the XOR operation 161 is performing in the background. If both the host bus 100 and the disk drive bus 120 request for cache data access from cache memory 130, time multiplex functions would be used to share the bus access. From the system or software point of view, the XOR operation 161, the host bus100 and the disk drive bus 120 accesses are concurrently happening.

As described above, Figure 2 shows the overall caching technique of Figure 1 and where the dual datapaths can fit in to further improve the caching technique.

To illustrate the dual datapath feature, when XOR 161 is in progress, and both the host bus 100 and the disk drive bus are trying to access the cache memory 130. With the dual datapath scheme in this architecture, the host bus and disk drive requests are queued and the two requests can indeed execute concurrently. Assume the host bus 100 asked for the transfer first and each bus 100, 120 requested for 64Kbyte transfer. The side bus interface 144 would divide each of the two transfers into smaller portion(assume 512Byte) and then interleave between the two buses 100, 120. In this case, the host bus 100 starts first since the first request was initialized by the host. The transfer begins with the first 512byte from the cache memory 130 to the host while the XOR 161 is halted. When this short transfer completed, the XOR logic 161 continues for 512Kbtye. Next, the disk drive request is granted. The transfer begins for the next 512byte from the cache memory 130 to the disk drive through disk drive bus 120. Once again the XOR 161 is put on hold. When the data is pumped out from the cache to the disk drive, the XOR 161 is activated again for another 512Kbtye. Once the 512byte operation is completed, the side bus interface 144 swings back to the host bus transfer. The same process continues until all 64Kbyte transfer is completed on both the host bus 100 and the disk drive bus 120. One may notice the cache bus is being multiplexed for the XOR function 161, the host bus 100 and disk drive bus 120. This capability allows the software to transfer data from/to the cache memory 130 while the lengthy XOR 161 is performing in the background. With this architecture, the data access from the cache memory 130 can be

performed without waiting for the completion of the XOR operation 161. Therefore, the overall system performance can be greatly improved.

Local Memory for Reducing Cache Memory Bus Traffic in RAID Application

As has been described above, by pulling the XOR function out of the RAID application software and applying caching technique on the host read and write operations, the overall system performance improves substantially.

When the XOR function implements in hardware and the data stores in the cache memory, the traffic on the memory bus becomes very heavily loaded while the XOR operation is being performed. Assume a RAID system with 4 data disks and 1 XOR disk, as shown in Figure 3. The cache memory 130 is partitioned into 5 logical units and assigned the units to be A unit, B unit, C unit, D unit, and X unit. Each unit A, B, C, D and X are assigned 64Kbytes and the host is storing data into the disk. Since there is a cache memory 130, the data will be storing into the cache memory 130 instead. The following are the sequences of operation to perform this task without any local memory.

1. write data to A unit
2. write data to B unit
3. write data to C unit
4. write data to D unit
5. read A unit and do XOR with zero
6. write result to X unit
7. read B unit
8. read X unit and do XOR with B
9. write result to X unit
10. read C unit
11. read X unit and do XOR with C
12. write result to X unit

13. read D unit
14. read X unit and do XOR with D
15. write result to X unit

If new data comes in to replace A unit, the following sequence of events will occur.

1. read A unit from cache
2. read X unit from cache and undo XOR with the old A
3. write result to X unit
4. write new data to A unit
5. read A unit from cache, the new data
6. read X unit and do XOR with the new A
7. write result to X unit

Each unit in the above example is 64Kbytes and this translated into many thousands of cycles depending on the implementation of the memory controller and the memory bus width.

In accordance with a further embodiment of the present invention, memory transfer cycles are reduced by a significant amount and reduce or save power at the same time due to less transaction on the bus. The technique is to add a local memory with the XOR hardware, as shown as block 160 in Figure 3. The incoming data from the cache memory 130 or from the external sources can perform XOR operation with the local memory data and the result stores back into the local memory 160. This basically accumulates the XOR result from all the desirable sources and then transfers this result to the cache memory 130 at one time. This effectively saves a lot of the intermediate transfer steps. The optimum size of the local memory is equal to the maximum host bus transfer size per transaction or commonly called strip/segment size. This size may vary

from one application to another application and the example above uses 64Kbytes per host transfer as a unit.

Walking through the same cases as the examples given above illustrates the improvement. When the host wants to store data to the disk, obviously data will go to the cache first and the sequence will be the following:

1. write data to A unit and do XOR with zero and put result in local memory
2. write data to B unit and do XOR with A and put result in local memory
3. write data to C unit and do XOR with A,B and put result in local memory
4. write data to D unit and do XOR with A,B,C and put result in local memory
5. write XOR result to X unit

This results in a total of five transfers compared with fifteen transfers previously and this means a 66% reduction in memory operations.

The other case when old data is replaced by new data in A unit, the following sequence of operations will be executed.

1. read A unit from cache and do XOR with zero
2. read X unit from cache, undo XOR from the old A data and store result in local memory
3. write new data to A unit and do XOR and store result in local memory
4. write XOR result to X unit

This results in a total of four transfers compared with seven transfers previously.

By having a local memory with the XOR function in hardware, this will save thousand of cycles on the cache memory bus and reduce power consumption from the components at the same time.

In Figures 3 and 4, with the local memory architecture, the intermediate XOR 166 result can be stored in the local memory 170 and the datapath can be arranged so that as the data comes-in from the local side bus 146, the data can go to the cache memory 130 by cache memory controller 159 and to the XOR 166 logic path as well. As mentioned above, this architecture not only can reduce the cache bandwidth by a substantial amount but also can complete the XOR transaction in less than half the time. The overhead of writing and reading to/from the cache memory 130 is totally eliminated and in addition, the XOR 166 function is performed as the data come-in from the local side bus 146. Since one source is ready in the local memory 170 and the other is coming in, the XOR 166 function is executed on the fly as the data flow through the datapath. The result is inputted back into the local memory 170 and it is ready to be used again for the next input source. With the local memory 170 embedded in this architecture clearly has demonstrated the effectiveness and the efficiency of enhancing the implementation of the XOR feature in the RAID application system.

The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and it should be understood that many modifications and variations are possible in light of the above teaching. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the Claims appended hereto and their equivalents.

WHAT IS CLAIMED IS:

1. A caching system for RAID applications, the system comprising:

a host bus;

a disk drive bus;

an internal cache memory;

interfacing means for interfacing between the host bus, the disk drive bus and the cache memory, the interfacing means including

a host interface for interfacing with the host bus;

a disk drive interface for interfacing with the disk drive bus;

XOR function means for interfacing with the cache memory and for providing and loading one or more XOR functions into the cache memory; and

a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory.

2. The system as in Claim 1 including DMA means for buffering the data transfer between the host bus, the disk drive bus and the cache memory.

3. The system as in Claim 2 including an internal side bus for interfacing with the XOR function means.

4. The system as in Claim 3 including means for providing direct data transfers between the host bus and the disk drive bus.

5. The system as in Claim 4 including means for providing simultaneous data transfers between the host bus and the disk drive bus and the cache memory.

6. A caching system comprising:

a host bus;

a disk drive bus;

an internal cache memory;

interfacing means for interfacing between the host bus, the disk drive bus and the cache memory, the interfacing means including

a host interface for interfacing with the host bus;

a disk drive interface for interfacing with the disk drive bus;

XOR function means for interfacing with the cache memory and for providing and loading one or more XOR functions into the cache memory; and

a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory.

7. A caching system comprising:

a host bus;

a disk drive bus;

an internal cache memory;

interfacing means for interfacing between the host bus, the disk drive bus and the cache memory, the interfacing means including

a host interface for interfacing with the host bus;

a disk drive interface for interfacing with the disk drive bus;

XOR function means for interfacing with the cache memory and for providing and loading one or more XOR functions into the cache memory; and

a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory;

DMA means for buffering the data transfer between the host bus, the disk drive bus and the cache memory;

an internal side bus for interfacing with the XOR function means;

means for providing direct data transfers between the host bus and the disk drive bus; and

means for providing simultaneous data transfers between the host bus and the disk drive bus and the cache memory.

8. A caching system for RAID applications, the system comprising:

a host bus;

a disk drive bus;

an internal cache memory;

interfacing means for interfacing between the host bus, the disk drive bus and the cache memory, the interfacing means including

a host interface for interfacing with the host bus;

a disk drive interface for interfacing with the disk drive bus;

dual XOR datapath function means for interfacing with the cache memory and for providing and loading one or more dual XOR datapath functions into the cache memory; and

a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory.

9. A caching system for RAID applications, the system comprising:

a host bus;

a disk drive bus;

an internal cache memory;

interfacing means for interfacing between the host bus, the disk drive bus and the cache memory, the interfacing means including

a host interface for interfacing with the host bus;

a disk drive interface for interfacing with the disk drive bus;

XOR function means with local memory for interfacing with the cache memory and for providing and loading one or more XOR functions with local controller into the cache memory; and

a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory.

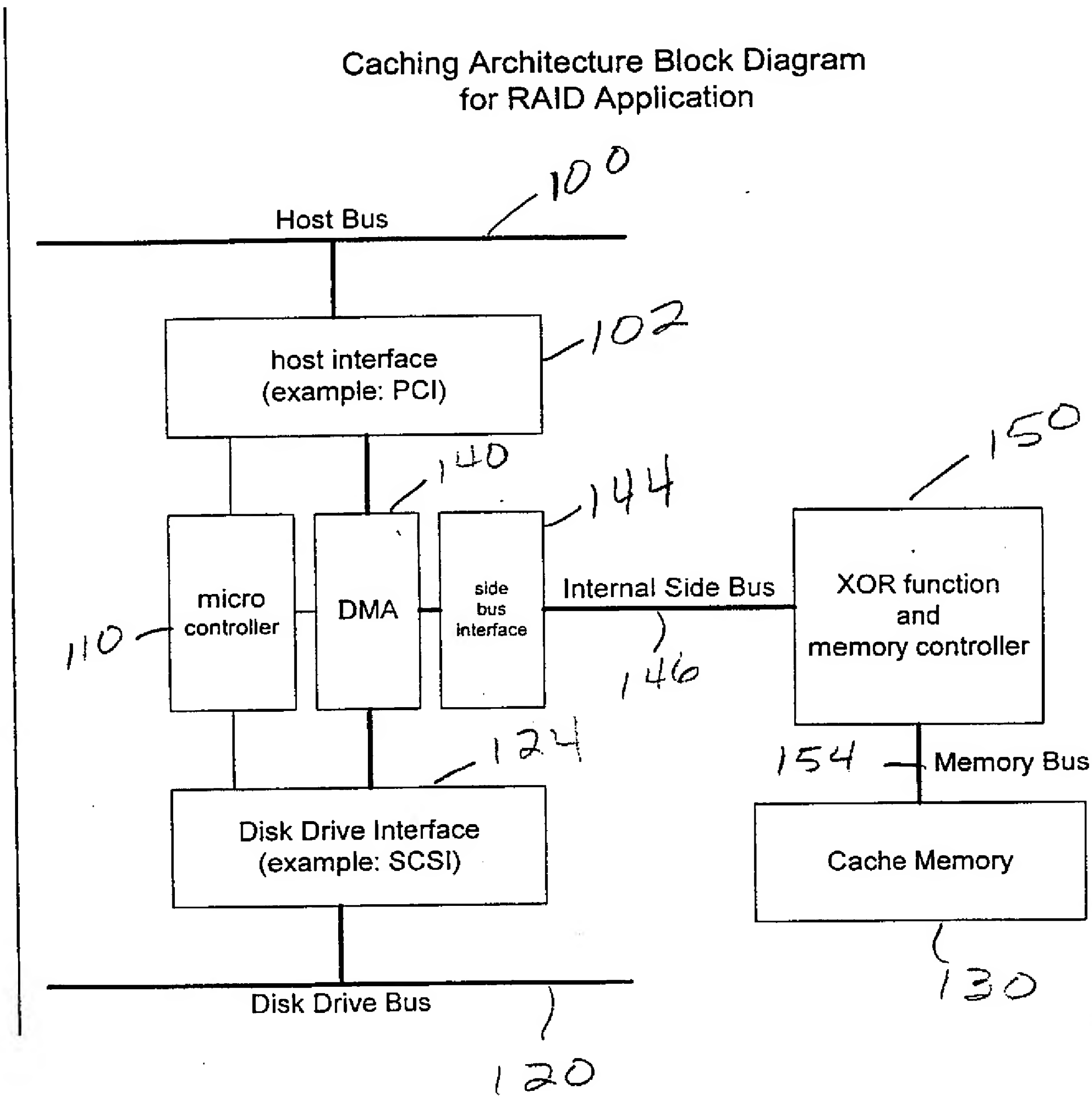

Caching Architecture Block Diagram
for RAID Application
100
Host Bus
host interface
(example: PCI)
102
140
144
150
micro
controller
110
DMA
side
bus
interface
Internal Side Bus
146
XOR function
and
memory controller
124
154
Memory Bus
Disk Drive Interface
(example: SCSI)
Cache Memory
130
Disk Drive Bus
120

FIG 1

Caching Architecture with embedded CPU
for RAID Application

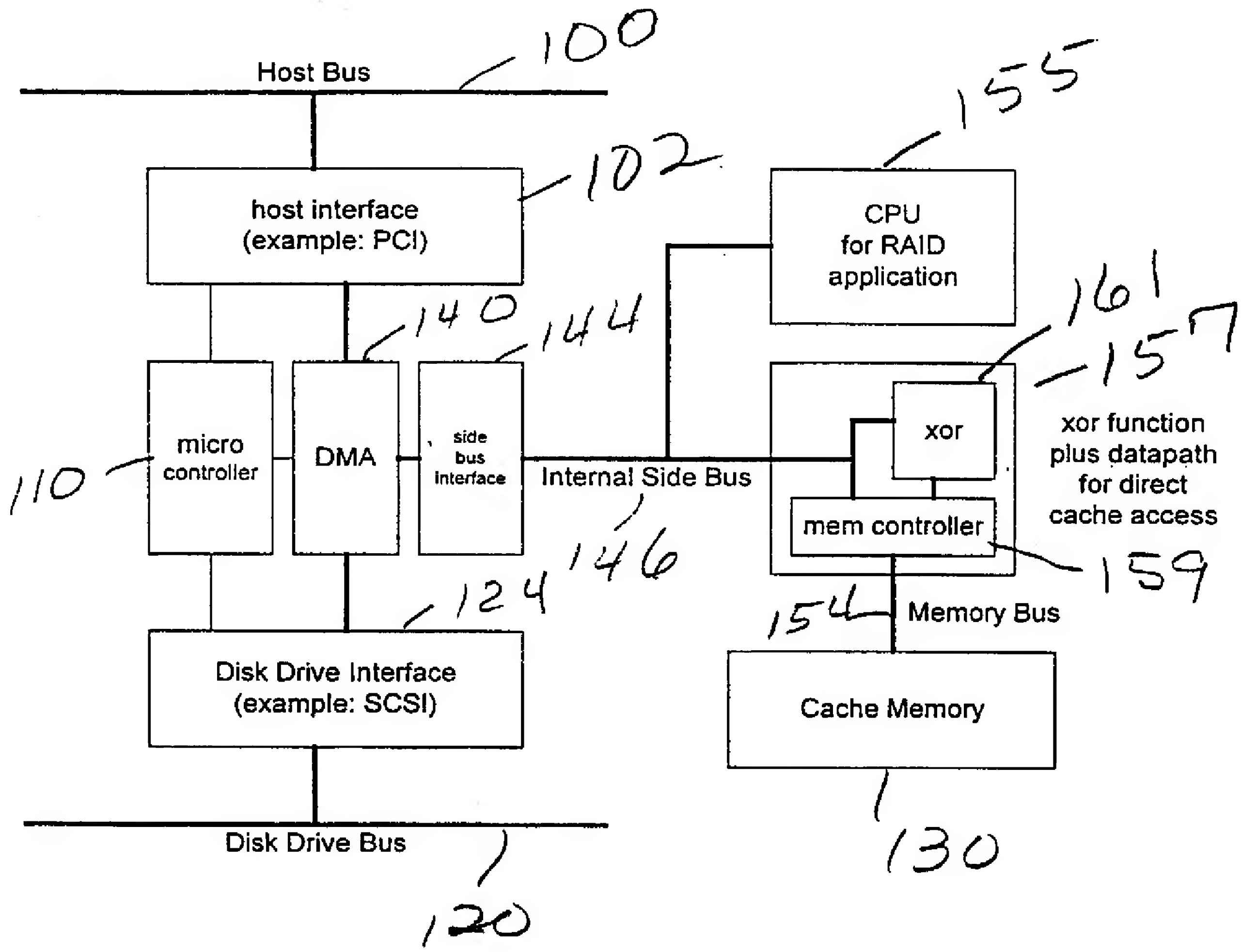


FIG 2

Caching Architecture Block Diagram
for RAID Application

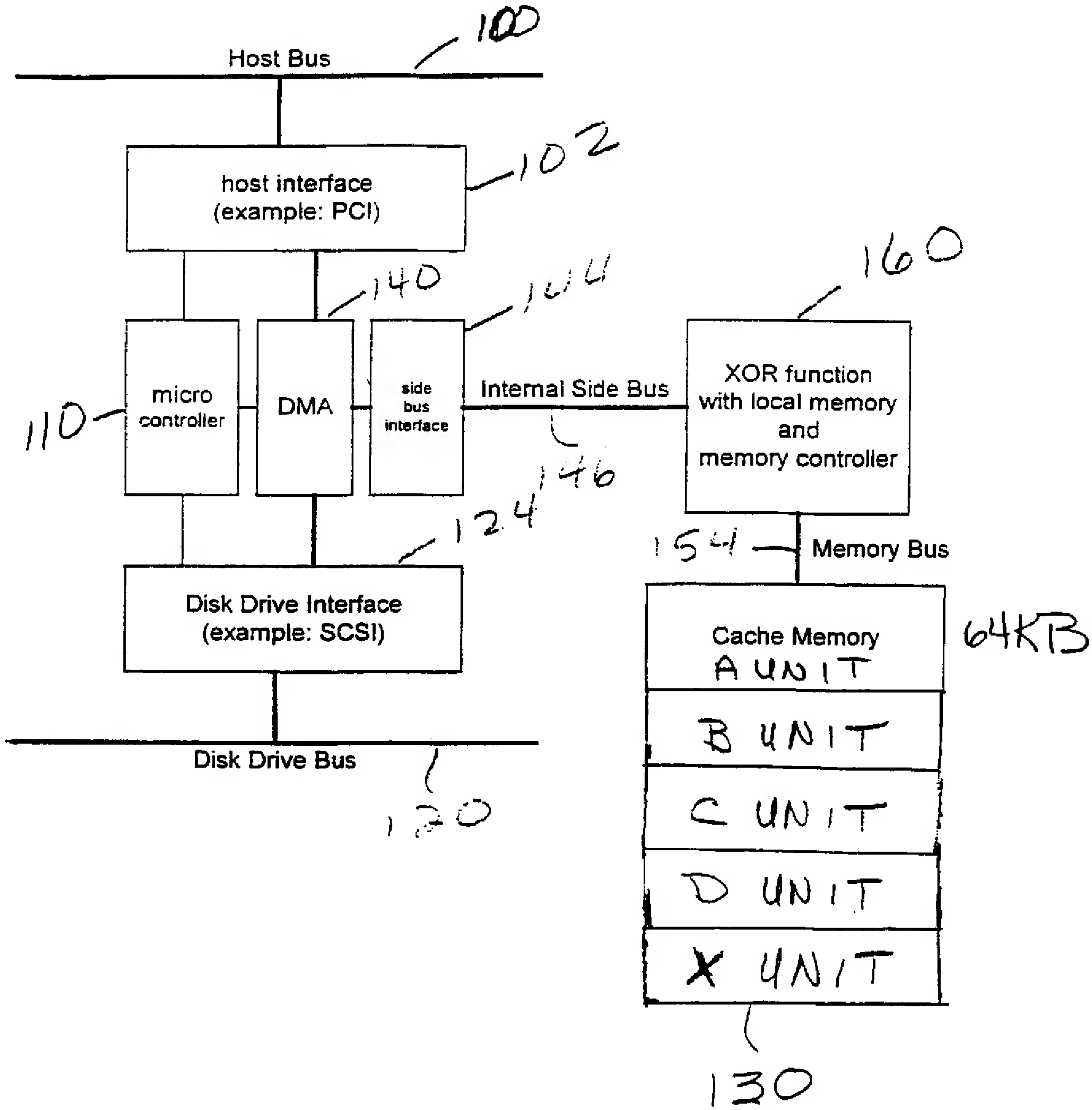


FIG 3

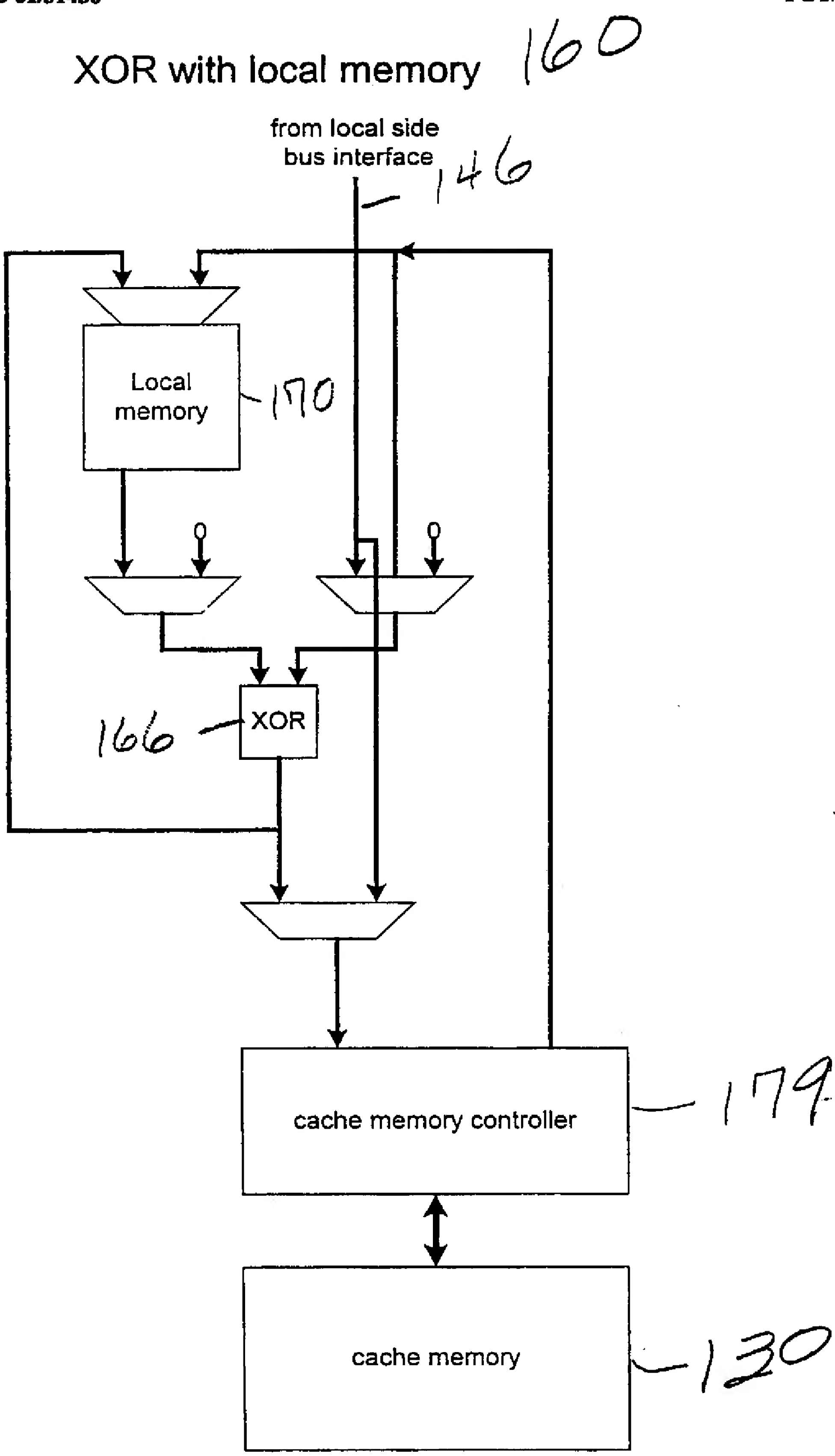

XOR with local memory
160
from local side
bus interface
146
Local
memory
170
0
0
166
XOR
cache memory controller
179
cache memory
130

FIG 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/29881

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 12/08

US CL :711/113, 114, 138, 168; 710/22

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 711/113, 114, 138, 168; 710/22

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X ---- Y	WILKES, JOHN et al., "The HP AutoRAID Hierarchical Storage System", Proceedings of the fifteenth ACM symposium on Operating systems principles; December 1995, pp. 96-108, especially figure 2 and page 101, left column, lines 8-14 and 21-27.	1-3, 6, 8 ------------ 4-5, 7, 9
X	MENON, JAI et al., "The Architecture of a Fault-Tolerant Cached RAID Controller", Proceedings of the 20th annual international symposium on Computer architecture; May 1993, pp. 76-86, especially figure 1 and page 78, right column, lines 24-25.	1, 6, 8

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
19 DECEMBER 2000	26 FEB 2001

Name and mailing address of the ISA/US	Authorized officer
Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	REGINALD G. BRAGDON Peggy Harrod
Facsimile No. (703) 305-3230	Telephone No. (703) 305-3823

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/29881

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	"Understanding HP AutoRAID Part II or II: The AutoRAID Storage Hierarchy Performance Considerations [online]. Enterprise Storage Solutions Division, Hewlett-Packard Company, March 1999 [retrieved from the internet 15 December 2000: <www.enterprisestorage.hp.com/products/disk_array/autoraid/sse_II_disk_array_12h_fa.html>]. See under "Cache Checkpointing".	5 and 7
Y	US 4,571,674 A (HARTUNG) 18 February 1986, col. 3, lines 31-32.	4-5 and 7
Y	US 5,206,943 A (CALLISON et al.) 27 April 1993, col. 2, lines 34-35 and 66-68, abstract, and figure 25B.	9
A	US 5, 522,065 A (NEUFELD) 28 May 1996, see figures 1 and 4.	9
A	US 5,937,174 A (WEBER) 10 August 1999, figures 2-4	1-3, 6, 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/29881

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

USPAT, EPO Abstracts, JPO Abstracts, Derwent, IBM TDB, IEEE Periodicals, ACM Periodicals
search terms: parity, xor, exclusive-or, cache, raid, disk, disc, dasd, (working or temp or temporary) adj (buffer or register)

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 May 2001 (03.05.2001)

PCT

(10) International Publication Number
WO 01/31456 A1

(51) International Patent Classification[7]: G06F 12/08

(21) International Application Number: PCT/US00/29881

(22) International Filing Date: 27 October 2000 (27.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/429,142 28 October 1999 (28.10.1999) US

(71) Applicant: CONNECTCOM SOLUTIONS, INC. [US/US]; 1150 Ringwood Court, San Jose, CA 95131 (US).

(72) Inventor: LAM, William; 757 Anacapa Court, Milpitas, CA 95953 (US).

(74) Agents: BALDWIN, Stephen, E. et al.; Trial & Technology Law Group, Suite 220, 545 Middlefield Road, Menlo Park, CA 94025 (US).

(81) Designated States (*national*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

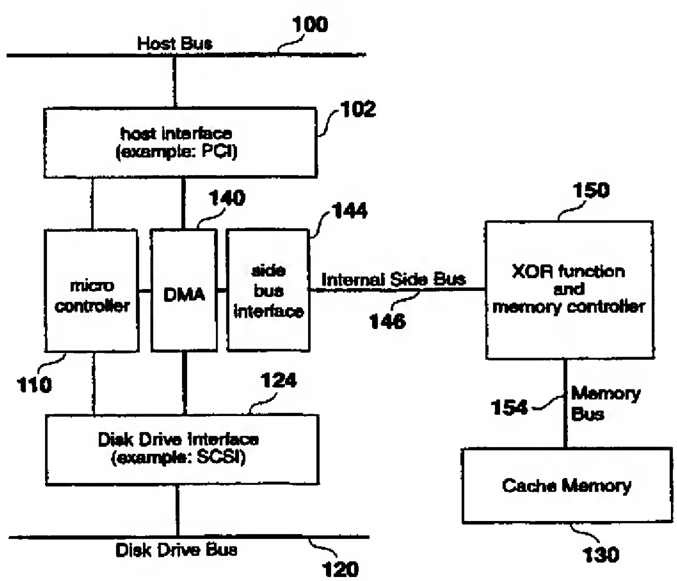
(84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

(48) Date of publication of this corrected version:
10 May 2002

[Continued on next page]

(54) Title: CACHING TECHNIQUES FOR IMPROVING SYSTEM PERFORMANCE IN RAID APPLICATIONS



(57) Abstract: The present invention provides a caching technique for improving system performance in RAID (Redundant Arrays of Inexpensive Disks) applications. The memory caching architecture technique according to the present invention creates a substantial improvement on the system host bus (100), especially for a RAID application system, implementing an Exclusive-OR (XOR) function in hardware. By pulling the XOR operation from software and by implementing the XOR function in hardware (150) and by removing the unnecessary data transfer from the XOR operation, a significant improvement on the host bus bandwith can be achieved.

WO 01/31456 A1

(15) Information about Correction:
see PCT Gazette No. 19/2002 of 10 May 2002, Section II

Caching Techniques for Improving System Performance in RAID Applications

By:
William Lam

Background of the Invention

The present invention relates to a caching technique for improving system performance in RAID applications. The memory caching architecture technique according to the present invention creates a substantial improvement on the system host bus, especially for a RAID application system.

The present invention is related to U. S. Patent Nos. 5,734,924 entitled System For Host Accessing Local Memory, issued March 31, 1998; 5,586,268 entitled Multiple Peripheral Adapter Device Driver Architecture, issued December 17, 1996; and 5,561,813 entitled Circuit For Resolving I/O Port Address Conflicts, issued October 1, 1996, all of which are assigned to the same assignee as the present invention, and the details of which are hereby incorporated by reference.

The traditional RAID (Redundant Arrays of Inexpensive Disks) architecture is to fetch data or to store data from/to SCSI Bus (a popular disk drive Bus) to/from PCI Bus (a popular system host Bus). In many situations, the system performance would be limited by the traffic on the Host Bus. A typical data transfer size from the Host to the disk drive is in the order of 4K byte to 64K byte. When the application software needs to perform the XOR operation (from the RAID algorithm), huge amounts of data transfer required to go on the host bus. In addition, each access from the hard disk drive is very

slow relative to silicon memory access time. This in term generates a bottleneck situation on the host bus which in-effect slows down the overall system performance.

Summary of the Invention

It is an object of the present invention to provide a caching technique for improving system performance in RAID applications.

The memory caching architecture technique according to the present invention creates a substantial improvement on the system host bus, especially for the RAID application system, implementing an Exclusive-OR (XOR) function in hardware. By pulling the XOR operation from software and by implementing the XOR function in hardware and by removing the unnecessary data transfer from the XOR operation, a significant improvement on the host bus bandwidth can be achieved.

In one embodiment, the present invention provides a caching system comprising a host bus; a disk drive bus; an internal cache memory; interfacing means for interfacing between the host bus, the disk drive bus and the cache memory. The interfacing means includes a host interface for interfacing with the host bus; a disk drive interface for interfacing with the disk drive bus; XOR function means for interfacing with the cache memory and for providing and loading one or more XOR functions into the cache memory; and a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory.

In a further embodiment, the present invention includes DMA means for buffering the data transfer between the host bus, the disk drive bus and the cache memory; an internal side bus for interfacing with the XOR function means for providing direct data transfers between the host bus and the disk drive bus; and means for providing simultaneous data transfers between the host bus and the disk drive bus and the cache memory.

Other objects, features and advantages of the present invention will become apparent from the following detailed description when taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings where like numerals indicate like components and which are incorporated in and form a part of this specification, illustrate embodiments of the invention and, together with the description, serve to explain the principles of the invention:

Figure 1 shows caching architecture block diagram for the present invention.

Figure 2 shows a caching architecture block diagram with a XOR datapath and with an embedded CPU for the present invention.

Figure 3 shows an XOR function with local memory for the present invention.

Figure 4 shows a more detailed XOR function with local memory of Figure 3.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. While the invention will be described in conjunction with the preferred embodiments, it will be understood that they are not intended to limit the invention to those embodiments. On the contrary, the invention is intended to cover alternatives, modifications and equivalents, which may be included within the spirit and scope of the invention as defined by the appended claims.

Caching Technique for RAID applications

As will be described, the memory caching architecture technique according to the present invention creates a substantial improvement on the system host bus, especially for the RAID application system, implementing an exclusive-OR (XOR) function in

hardware. By pulling the XOR operation from software and by implementing the XOR function in hardware and by removing the unnecessary data transfer from the XOR operation, a significant improvement on the host bus bandwidth can be achieved.

The present invention introduces a caching technique which resides between the disk drive bus and the host bus. By having a local memory sitting between the Host Bus and the disk drive Bus, the data access time from the host can be significantly reduced when there is a cache hit or the data is in the cache memory. For the case when data is not in the cache memory, data is pulled out from the disk to the cache memory and to the host bus. A read ahead technique can be used in this case as well. More data is read out than required, which translates into more chances for the host to have a cache hit, perhaps for the next time around.

Even when the host performs a write operation to the disk, the data transfers to the cache first and then releases the host bus for other operations. This accelerates the termination of the host bus write cycle and the write transfer from the cache to the disk drive performed by the controller locally at a later time. This would isolate the host bus from the slow disk drive. The minimum size of the cache is equal to the maximum amount of each data transfer from the host multiply by the total number of the disk drives.

Since the XOR operation is extracted out from the software application, the host is no longer necessary to access the data from the slow disk drive bus to the host bus. The XOR operation can be performed behind the scene and the RAID software just dispatches the XOR operation to the bus controller. The heavy data accesses between the disk drive and the cache memory are totally isolated from the host bus. This boosts up the overall host bus performance and reduces the software time to perform the XOR operation.

Figure 1 shows the architecture described above.

Figure 1 is one example of the RAID application architecture for an embedded system. In Figure 1, the host bus interface block 102 provides the central communication between the host bus 100 and the host adapter circuitry. In most systems today, the host bus 100 would be a PCI bus. In this application, the host bus 100 communicates with the hard disk drive bus 120 through disk drive interface 124 through this architecture. This architecture provides direct data transfer from the host bus 100 to the disk drive via disk drive bus 120 or simultaneously transfer from the disk drive bus 120 to both the host bus 100 and to the cache memory 130.

This data transfer can be achieved by using the DMA datapath 140 which resides at the center of the three buses. The DMA (Direct Memory Access) datapath 140 is a media to buffer up the incoming data and to pump the data out at the appropriate time, especially since the speed from each of the three buses may not be the same. The data flow and traffic between the buses is managed by the micro-controller 110.

With the proper programming on the micro-controller 110, an optimized data flow can be precisely controlled, the caching technique can be realized, and the XOR function can be performed. The micro-controller 110 accepts the appropriate command blocks from the RAID software and the micro-controller 110 ucode would decode and execute the commands in a way such that the controls are enabling the proper datapath for a specific task. The DMA datapath 140 would provide enough data buffering on each side of the three buses, so that data transfer can be operated in an effective manner. The proper buffering size would determine by the typical transferring size, the incoming data rate and the outgoing data rate. With this architecture, the cache data can transfer to/from the host bus and the disk drive bus concurrently. This effect can be achieved from the side bus interface 144 by providing dual datapath from side bus 146 to host bus 100 and to hard disk bus 120. By alternating the transfer to this two datapath in a reasonable transfer size(such as 512B each time), a concurrent transferring effect from the software point of view can be accomplished.

One bus that DMA 140 interfaces with is the side bus 146 which communicates with the cache controller and the XOR functional block 150. This block 150 provides the capability of doing an XOR function and interfacing with the cache memory 130 via memory bus 154. This capability enables the RAID application to substantially improve the system performance. By performing the XOR function on the side while freeing up the host bus 100 for other tasks, the host bus bandwidth is improved significantly and at the same time, the time for performing the XOR task is reduced.

The side bus 146 accepts multiple XOR blocks. In this way, the expandable cache memory and multiple XOR functions can execute simultaneously. For the high-end server class system, bigger cache size and multiple XOR functions actively performing simultaneously would be desirable.

To illustrate this architecture without a microprocessor, a 64Mbyte transfer is activated from disk drive bus 120 to host bus 100 via host interface 102. With this architecture, the data transfers from the disk drive bus 120 to the host bus 100, and at the same time, the same data transfers into the cache memory 130 as well. When the host requires fetching the same set of data, the data can be fed to the host as fast as several hundreds of nanoseconds. Since the data resides in the cache memory 150, the data can be quickly accessed without going through the hard disk which normally would take up to few milliseconds. To perform the XOR function, the RAID software loads the proper source data into the cache memory 130, and the RAID software would execute the command by properly programmed the internal required registers.

The hardware puts the XOR result into the cache memory 130, and the RAID software gets the data from the cache memory 130 and put this result into the disk drive. During the XOR operation, there is no host bus transfer required, and as a result the host bus 100 is freed up during this lengthy operation.

With a microprocessor 155 embedded into the architecture as shown in Figure 1B, the top level command receives from the host and the microprocessor runs the RAID

software to decode the command and formed a list of lower level commands which usually send over the host bus 100. Now these lower level commands are sent to the micro-controller for the next level of execution through the internal bus 146. In Figure 2, the XOR function plus datapath 157 for direct cache memory 130 access includes XOR 161 and memory controller 159. This method will even further reduce the traffic on the host interface bus 102. With this embedded microprocessor capability, a complete solution for RAID application is available on a single component and many RAID systems can now be built in a cost effective manner.

Dual Data Path Providing Concurrent Cache Data Access and XOR operation

As described above, by pulling the XOR function out of the RAID application software and applying caching technique on the host read and write operations, the overall system performance improves substantially.

While the XOR hardware is executing, the data in the cache memory is locking up exclusively for this operation. The XOR operation can be very lengthy in time depending on the number of the input sources and the size of the data. If there are 8 input sources and each source is 64Kbytes, the XOR operation would occupy the cache data for few mini-seconds. This is a very long period of time. If the host bus or the disk drive bus needs to access the cache data, there is a long waiting period before the cache memory is free for access.

In accordance with a further embodiment of the present invention, the technique to improve this situation is to implement a dual datapath function 157, as shown in Figure 2. One datapath is used for the XOR operation and the other is used for direct cache memory access. While the XOR datapath 161 is performing the operation, the host bus 100 or the disk drive bus 120 can access the cache data through the second datapath. This technique would delay the XOR operation from finishing but the overall performance would improve. Since neither the host bus 100 nor the disk drive bus 120 need not have

to wait for the XOR operation 161 to complete, the RAID application software can continue to process while the XOR operation 161 is performing in the background. If both the host bus 100 and the disk drive bus 120 request for cache data access from cache memory 130, time multiplex functions would be used to share the bus access. From the system or software point of view, the XOR operation 161, the host bus100 and the disk drive bus 120 accesses are concurrently happening.

As described above, Figure 2 shows the overall caching technique of Figure 1 and where the dual datapaths can fit in to further improve the caching technique.

To illustrate the dual datapath feature, when XOR 161 is in progress, and both the host bus 100 and the disk drive bus are trying to access the cache memory 130. With the dual datapath scheme in this architecture, the host bus and disk drive requests are queued and the two requests can indeed execute concurrently. Assume the host bus 100 asked for the transfer first and each bus 100, 120 requested for 64Kbyte transfer. The side bus interface 144 would divide each of the two transfers into smaller portion(assume 512Byte) and then interleave between the two buses 100, 120. In this case, the host bus 100 starts first since the first request was initialized by the host. The transfer begins with the first 512byte from the cache memory 130 to the host while the XOR 161 is halted. When this short transfer completed, the XOR logic 161 continues for 512Kbtye. Next, the disk drive request is granted. The transfer begins for the next 512byte from the cache memory 130 to the disk drive through disk drive bus 120. Once again the XOR 161 is put on hold. When the data is pumped out from the cache to the disk drive, the XOR 161 is activated again for another 512Kbtye. Once the 512byte operation is completed, the side bus interface 144 swings back to the host bus transfer. The same process continues until all 64Kbyte transfer is completed on both the host bus 100 and the disk drive bus 120. One may notice the cache bus is being multiplexed for the XOR function 161, the host bus 100 and disk drive bus 120. This capability allows the software to transfer data from/to the cache memory 130 while the lengthy XOR 161 is performing in the background. With this architecture, the data access from the cache memory 130 can be

performed without waiting for the completion of the XOR operation 161. Therefore, the overall system performance can be greatly improved.

Local Memory for Reducing Cache Memory Bus Traffic in RAID Application

As has been described above, by pulling the XOR function out of the RAID application software and applying caching technique on the host read and write operations, the overall system performance improves substantially.

When the XOR function implements in hardware and the data stores in the cache memory, the traffic on the memory bus becomes very heavily loaded while the XOR operation is being performed. Assume a RAID system with 4 data disks and 1 XOR disk, as shown in Figure 3. The cache memory 130 is partitioned into 5 logical units and assigned the units to be A unit, B unit, C unit, D unit, and X unit. Each unit A, B, C, D and X are assigned 64Kbytes and the host is storing data into the disk. Since there is a cache memory 130, the data will be storing into the cache memory 130 instead. The following are the sequences of operation to perform this task without any local memory.

1. write data to A unit
2. write data to B unit
3. write data to C unit
4. write data to D unit
5. read A unit and do XOR with zero
6. write result to X unit
7. read B unit
8. read X unit and do XOR with B
9. write result to X unit
10. read C unit
11. read X unit and do XOR with C
12. write result to X unit

13. read D unit
14. read X unit and do XOR with D
15. write result to X unit

If new data comes in to replace A unit, the following sequence of events will occur.

1. read A unit from cache
2. read X unit from cache and undo XOR with the old A
3. write result to X unit
4. write new data to A unit
5. read A unit from cache, the new data
6. read X unit and do XOR with the new A
7. write result to X unit

Each unit in the above example is 64Kbytes and this translated into many thousands of cycles depending on the implementation of the memory controller and the memory bus width.

In accordance with a further embodiment of the present invention, memory transfer cycles are reduced by a significant amount and reduce or save power at the same time due to less transaction on the bus. The technique is to add a local memory with the XOR hardware, as shown as block 160 in Figure 3. The incoming data from the cache memory 130 or from the external sources can perform XOR operation with the local memory data and the result stores back into the local memory 160. This basically accumulates the XOR result from all the desirable sources and then transfers this result to the cache memory 130 at one time. This effectively saves a lot of the intermediate transfer steps. The optimum size of the local memory is equal to the maximum host bus transfer size per transaction or commonly called strip/segment size. This size may vary

from one application to another application and the example above uses 64Kbytes per host transfer as a unit.

Walking through the same cases as the examples given above illustrates the improvement. When the host wants to store data to the disk, obviously data will go to the cache first and the sequence will be the following:

1. write data to A unit and do XOR with zero and put result in local memory
2. write data to B unit and do XOR with A and put result in local memory
3. write data to C unit and do XOR with A,B and put result in local memory
4. write data to D unit and do XOR with A,B,C and put result in local memory
5. write XOR result to X unit

This results in a total of five transfers compared with fifteen transfers previously and this means a 66% reduction in memory operations.

The other case when old data is replaced by new data in A unit, the following sequence of operations will be executed.

1. read A unit from cache and do XOR with zero
2. read X unit from cache, undo XOR from the old A data and store result in local memory
3. write new data to A unit and do XOR and store result in local memory
4. write XOR result to X unit

This results in a total of four transfers compared with seven transfers previously.

By having a local memory with the XOR function in hardware, this will save thousand of cycles on the cache memory bus and reduce power consumption from the components at the same time.

In Figures 3 and 4, with the local memory architecture, the intermediate XOR 166 result can be stored in the local memory 170 and the datapath can be arranged so that as the data comes-in from the local side bus 146, the data can go to the cache memory 130 by cache memory controller 159 and to the XOR 166 logic path as well. As mentioned above, this architecture not only can reduce the cache bandwidth by a substantial amount but also can complete the XOR transaction in less than half the time. The overhead of writing and reading to/from the cache memory 130 is totally eliminated and in addition, the XOR 166 function is performed as the data come-in from the local side bus 146. Since one source is ready in the local memory 170 and the other is coming in, the XOR 166 function is executed on the fly as the data flow through the datapath. The result is inputted back into the local memory 170 and it is ready to be used again for the next input source. With the local memory 170 embedded in this architecture clearly has demonstrated the effectiveness and the efficiency of enhancing the implementation of the XOR feature in the RAID application system.

The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and it should be understood that many modifications and variations are possible in light of the above teaching. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the Claims appended hereto and their equivalents.

WHAT IS CLAIMED IS:

1. A caching system for RAID applications, the system comprising:

a host bus;

a disk drive bus;

an internal cache memory;

interfacing means for interfacing between the host bus, the disk drive bus and the cache memory, the interfacing means including

a host interface for interfacing with the host bus;

a disk drive interface for interfacing with the disk drive bus;

XOR function means for interfacing with the cache memory and for providing and loading one or more XOR functions into the cache memory; and

a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory.

2. The system as in Claim 1 including DMA means for buffering the data transfer between the host bus, the disk drive bus and the cache memory.

3. The system as in Claim 2 including an internal side bus for interfacing with the XOR function means.

4. The system as in Claim 3 including means for providing direct data transfers between the host bus and the disk drive bus.

5. The system as in Claim 4 including means for providing simultaneous data transfers between the host bus and the disk drive bus and the cache memory.

6. A caching system comprising:

a host bus;

a disk drive bus;

an internal cache memory;

interfacing means for interfacing between the host bus, the disk drive bus and the cache memory, the interfacing means including

a host interface for interfacing with the host bus;

a disk drive interface for interfacing with the disk drive bus;

XOR function means for interfacing with the cache memory and for providing and loading one or more XOR functions into the cache memory; and

a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory.

7. A caching system comprising:

a host bus;

a disk drive bus;

an internal cache memory;

interfacing means for interfacing between the host bus, the disk drive bus and the cache memory, the interfacing means including

a host interface for interfacing with the host bus;

a disk drive interface for interfacing with the disk drive bus;

XOR function means for interfacing with the cache memory and for providing and loading one or more XOR functions into the cache memory; and

a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory;

DMA means for buffering the data transfer between the host bus, the disk drive bus and the cache memory;

an internal side bus for interfacing with the XOR function means;

means for providing direct data transfers between the host bus and the disk drive bus; and

means for providing simultaneous data transfers between the host bus and the disk drive bus and the cache memory.

8. A caching system for RAID applications, the system comprising:
a host bus;
a disk drive bus;
an internal cache memory;
interfacing means for interfacing between the host bus, the disk drive bus and the cache memory, the interfacing means including

a host interface for interfacing with the host bus;

a disk drive interface for interfacing with the disk drive bus;

dual XOR datapath function means for interfacing with the cache memory and for providing and loading one or more dual XOR datapath functions into the cache memory; and

a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory.

9. A caching system for RAID applications, the system comprising:
a host bus;
a disk drive bus;
an internal cache memory;
interfacing means for interfacing between the host bus, the disk drive bus and the cache memory, the interfacing means including

a host interface for interfacing with the host bus;

a disk drive interface for interfacing with the disk drive bus;

XOR function means with local memory for interfacing with the cache memory and for providing and loading one or more XOR functions with local controller into the cache memory; and

a micro-controller means for controlling data flow between the host bus, the disk drive bus and the cache memory, including the XOR functions from the cache memory.

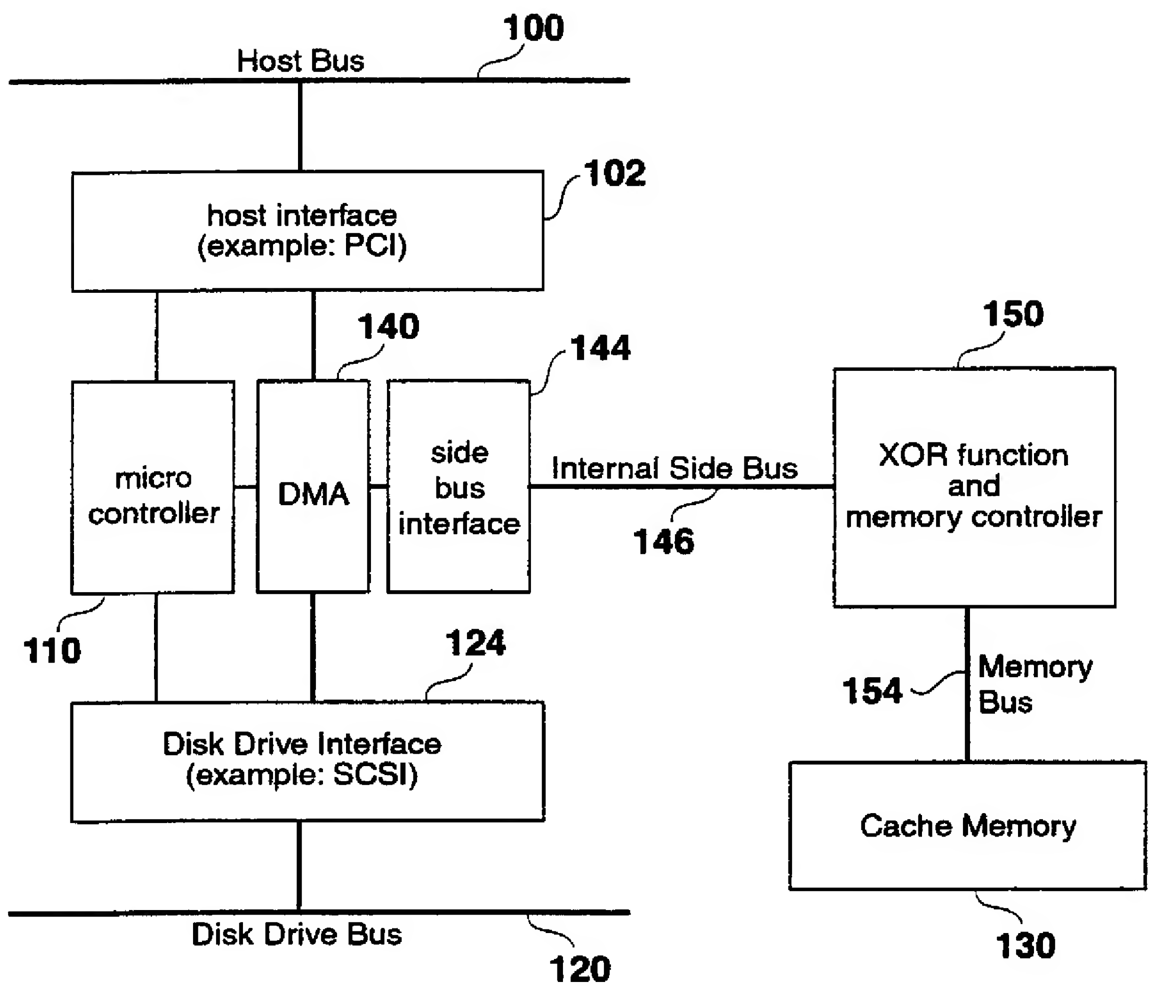

100
Host Bus
102
host interface
(example: PCI)
140
144
150
micro
controller
DMA
side
bus
interface
Internal Side Bus
XOR function
and
memory controller
146
110
124
154
Memory
Bus
Disk Drive Interface
(example: SCSI)
Cache Memory
Disk Drive Bus
120
130

Fig. 1

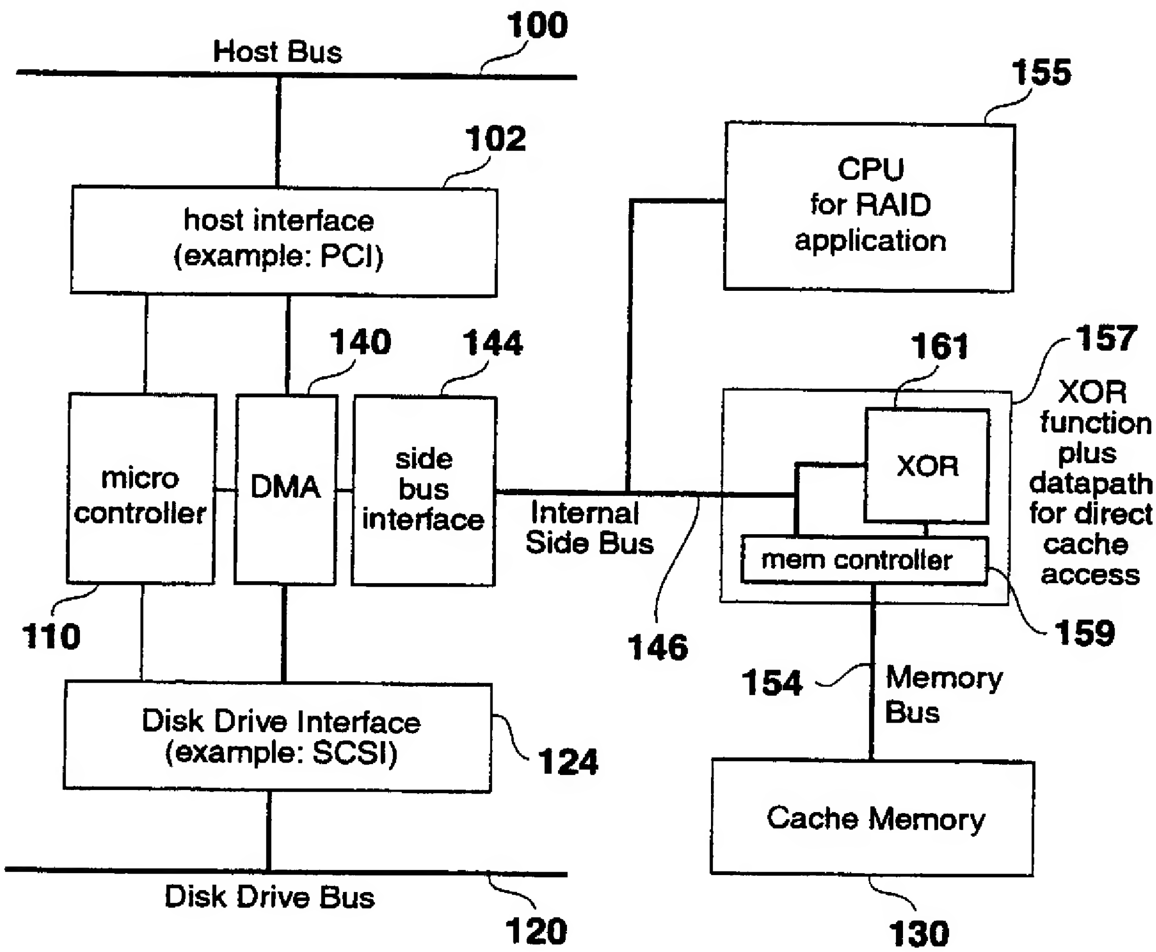


Fig. 2

 SUBSTITUTE SHEET (RULE 26)

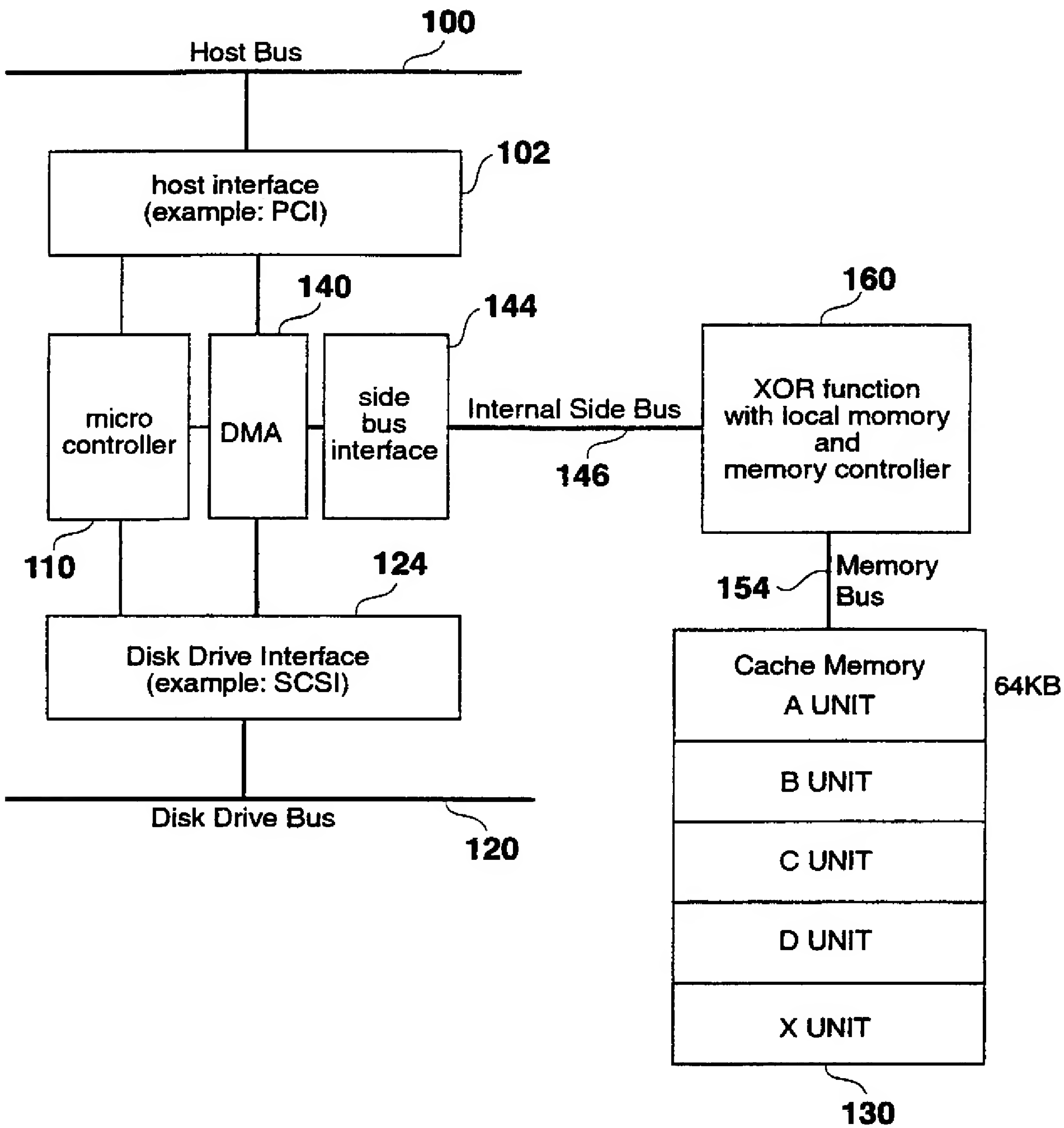

100
Host Bus
102
host interface
(example: PCI)
140
144
160
micro
controller
DMA
side
bus
interface
Internal Side Bus
146
XOR function
with local momory
and
memory controller
110
124
154
Memory
Bus
Disk Drive Interface
(example: SCSI)
Cache Memory
A UNIT
64KB
B UNIT
C UNIT
D UNIT
X UNIT
Disk Drive Bus
120
130

Fig. 3

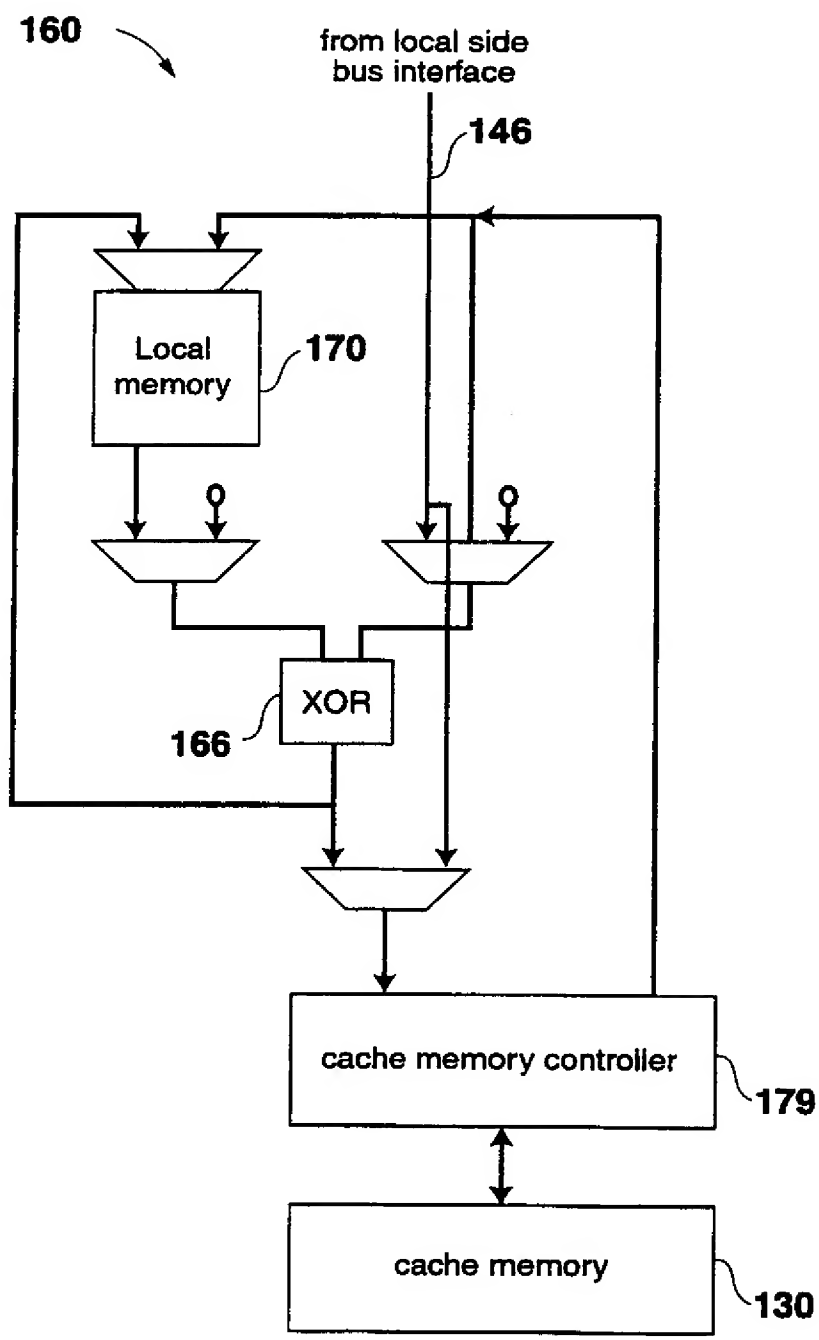

160
from local side
bus interface
146
Local
memory
170
0
0
XOR
166
cache memory controller
179
cache memory
130

Fig. 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/29881

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 12/08
US CL :711/113, 114, 138, 168; 710/22
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 711/113, 114, 138, 168; 710/22

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X ---- Y	WILKES, JOHN et al., "The HP AutoRAID Hierarchical Storage System", Proceedings of the fifteenth ACM symposium on Operating systems principles; December 1995, pp. 96-108, especially figure 2 and page 101, left column, lines 8-14 and 21-27.	1-3, 6, 8 ------------ 4-5, 7, 9
X	MENON, JAI et al., "The Architecture of a Fault-Tolerant Cached RAID Controller", Proceedings of the 20th annual international symposium on Computer architecture; May 1993, pp. 76-86, especially figure 1 and page 78, right column, lines 24-25.	1, 6, 8

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search 19 DECEMBER 2000	Date of mailing of the international search report 26 FEB 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer REGINALD G. BRAGDON Peggy Harrod Telephone No. (703) 305-3823

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	"Understanding HP AutoRAID Part II or II: The AutoRAID Storage Hierarchy Performance Considerations [online]. Enterprise Storage Solutions Division, Hewlett-Packard Company, March 1999 [retrieved from the internet 15 December 2000: <www.enterprisestorage.hp.com/products/disk_array/autoraid/sse_II_disk_array_12h_fa.html>]. See under "Cache Checkpointing".	5 and 7
Y	US 4,571,674 A (HARTUNG) 18 February 1986, col. 3, lines 31-32.	4-5 and 7
Y	US 5,206,943 A (CALLISON et al.) 27 April 1993, col. 2, lines 34-35 and 66-68, abstract, and figure 25B.	9
A	US 5, 522,065 A (NEUFELD) 28 May 1996, see figures 1 and 4.	9
A	US 5,937,174 A (WEBER) 10 August 1999, figures 2-4	1-3, 6, 8

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

USPAT, EPO Abstracts, JPO Abstracts, Derwent, IBM TDB, IEEE Periodicals, ACM Periodicals
search terms: parity, xor, exclusive-or, cache, raid, disk, disc, dasd, (working or temp or temporary) adj (buffer or register)